A RAND NOTE

DISTRIBUTED PROBLEM SOLVING FOR AIR FLEET CONTROL: FRAMEWORK AND IMPLEMENTATIONS

R. Steeb, D. McArthur, S. Cammarata,
S. Narain, W. Giarla

April 1984

N-2139-ARPA

Prepared for

The Defense Advanced Research Projects Agency

Rand
1700 MAIN STREET
P.O. BOX 2138
SANTA MONICA, CA 90406-2138

The research described in this report was sponsored by the Defense Advanced Research Projects Agency under ARPA Order No. 3460, Contract No. MDA903-82-C-0061, Information Processing Techniques Office.

Published by The Rand Corporation

A RAND NOTE

DISTRIBUTED PROBLEM SOLVING FOR AIR FLEET CONTROL: FRAMEWORK AND IMPLEMENTATIONS

R. Steeb, D. McArthur, S. Cammarata,
S. Narain, W. Giarla

April 1984

N-2139-ARPA

Prepared for

The Defense Advanced Research Projects Agency

Rand
1700 MAIN STREET
P.O. BOX 2138
SANTA MONICA, CA 90406-2138

PREFACE

This Note reports the interim results of an ongoing investigation of distributed problem solving for air fleet control, conducted for the Information Processing Techniques Office, Defense Advanced Research Projects Agency. The work has focused on the development of organizational structures for cooperative planning in complex, spatially distributed systems, using air-traffic control and remotely piloted vehicle (RPV) fleet control as illustrative contexts. Related research is reported in the following Rand publications:

Distributed Intelligence for Air Fleet Control, by R. Steeb, S. Cammarata, F. A. Hayes-Roth, P. W. Thorndyke, and R. B. Wesson, R-2728-ARPA, October 1981.

The ROSS Language Manual, by D. McArthur and P. Klahr, N-1854-AF, September 1982.

Swirl: Simulating Warfare in the ROSS Language, by P. Klahr, D. McArthur, S. Narain, and E. Best, N-1885-AF, September 1982.

Strategies of Cooperation in Distributed Problem Solving, by S. Cammarata, D. McArthur, and R. Steeb, N-2031-ARPA, December 1983.

SUMMARY

Distributed problem solving, or multiple-agent problem solving, refers to the process by which several agents interact to achieve goals. In this Note, we describe the development of a framework for implementation of multiple cooperative agents. We also describe experiments and demonstrations with different strategies of cooperation, using air-traffic control and remotely piloted vehicle (RPV) fleet coordination as our exemplary task domains.

Multiagent cooperation is discussed first in a domain-independent fashion, and then in the context of the two task domains. We contrast the methodologies, difficulties, and opportunities of distributed and centralized problem solving. From this analysis, we postulate a set of requirements on the information-gathering and organizational policies of group problem-solving agents, and we develop a general framework for implementing such policies. We then discuss a set of distributed problem solvers that we have developed for air-traffic control and surveillance RPV fleet control. Finally, we describe some experimental findings using the cooperative strategies, with particular emphasis on role assignment within the group and communications between group members.

CONTENTS

FIGURES

TABLE

I. INTRODUCTION

Distributed problem solving, or multiple-agent problem solving, refers to the process by which several agents interact to achieve goals. The intent of a theory of distributed problem solving is to develop an information processing account of effective group problem-solving performance. We are attempting to learn how groups can cooperate effectively by describing in formal, computational terms the actions of each agent as the group achieves a collective goal.

Previous work in cognitive science helps little in achieving this understanding. Few studies have focused on group problem solving. Over the past twenty years, beginning with the pioneering work of Newell and Simon, cognitive scientists have learned much about the information processing that underlies the problem solving of individuals. Cognitive psychologists have, for example, carefully studied the way in which people play games, solve mathematics problems, and program computers. In a similar way, workers in artificial intelligence have developed computational models of how single agents might construct blocks-world artifacts, do medical diagnosis, and plan genetics experiments, to mention only a few. These efforts have resulted in the development of a variety of techniques for modeling the environment, planning under uncertainty, and executing complex sequences of actions. Unfortunately, recent work suggests that the representations of knowledge (Konolige, 1981; Appelt, 1982) and planning expertise (McArthur and Klahr, 1982) required of agents in distributed or group problem-solving situations are quite different than those required for single-agent problem solvers. Organizational psychologists have explicitly studied group performance (Dalkey, 1977), but because of the difficulties of representing multiple disparate world views and of specifying sequences of activities within and between agents, their theories are usually expressed informally. Also, the theories are typically stated in aggregate terms, not in terms of the information processing of individual agents. The related area of distributed processing has more formal underpinnings but is also of limited applicability. In

distributed processing, multiple computers interact in a relatively simple fashion through the sharing of data. In distributed problem solving, agents must not only share data, but they must also *share the problem solving*.

In this Note, we present a model of distributed problem-solving processes. Our approach has been to first carefully study the competences or capabilities of agents in groups, then develop a computational model of how these capabilities might be achieved. Then we proceed to implement the computational theory and to develop and test specific distributed problem-solving systems. Section II discusses the difficulties and opportunities facing multiple-agent problem solvers in many domains and contrasts these domains with more frequently studied single-agent problem-solving environments. Section III uses this analysis to infer a set of requirements on the cooperative strategies of group problem-solving agents. In Sec. IV, we discuss the computational theory that follows from our analysis of competences. The remaining sections are devoted to descriptions of several specific systems we have implemented in the domains of distributed air-traffic control and distributed RPV coordination.

II. DISTRIBUTED AND SINGLE-AGENT PROBLEM SOLVING

To understand the capabilities of agents that solve problems in a distributed fashion, and to understand how they differ from single-agent problem solvers, we begin by examining some important characteristics of distributed problems.

CHARACTERIZATION OF DISTRIBUTED PROBLEM SOLVING

Several general characteristics of distributed problem-solving situations are particularly important for our purposes:

- Most situations consist of a collection of agents, each with various *skills*, including sensing, communication (often over limited-bandwidth channels), planning, and acting.
- The group as a whole has a set of assigned *tasks*. As in single-agent problem-solving situations, these tasks may need to be decomposed into subtasks, not all of which may be logically independent. The group must somehow assign subtasks to appropriate agents.
- Each agent typically has only *limited knowledge*. An agent may be subject to several kinds of limitations: limited knowledge of the environment (e.g., because of restricted sensing horizons), limited knowledge of the tasks of the group, or limited knowledge of the intentions of other agents.
- There are often limited shared *resources* that each agent can apply to tasks. For example, if the agents are in a blocks-world environment, the shared resources are the blocks out of which their constructions must be made.
- Agents typically have differing *appropriateness* for a given task. The appropriateness of a particular agent for a task is a function of how well the agent's skills match the expertise required to do the task, the extent to which its limited knowledge is adequate for the task, its current processing resources, and the quality of its communication links with other agents.

DIFFICULTIES IN DISTRIBUTED PROBLEM SOLVING

Several difficulties can arise when solving problems in a distributed fashion that are not significant in most single-agent problem-solving situations. First, in single-agent problem solving, the agent is typically *given* its task as part of the problem definition (Sacerdoti, 1974; Fahlman, 1974), whereas in distributed situations, the *assignment* of tasks to the agents is part of the group problem-solving activity. This assignment can be challenging. Many mappings of tasks to agents are possible, but because agents typically have differing available expertise for a given task, only a few agents will be acceptable for each task. Thus in many distributed problems, it is crucial for agents to adopt the right *role*. It would not be reasonable to assign the role of inventing a new chip to a lawyer, or the role of writing the patent to an engineer. In addition to ensuring that each task is assigned to an acceptable agent, the group has to ensure *task coverage*. Specifically, this means that all tasks should be assigned to some agent (complete role assignment) and that extra or redundant agents should not be assigned tasks (consistent role assignment). For example, in air-traffic control, if the task is to solve a possible spatial conflict, it may be critical to ensure that only one aircraft detours; if two or more adopt that role, they may possibly create a new collision situation.

Compounding the difficulty of finding an optimal task assignment is the limited knowledge of the agents. In most single-agent problem solvers, the agent has a complete world model, which usually remains complete because (1) all changes in the environment are made by the agent and thus it can always update its world model, and (2) a single agent does not have to worry about unknown intentions of other agents. The incomplete or incorrect world models of distributed agents may degrade the effectiveness of task assignment, either because agents that know the breakdown of tasks may not know which agents have the most appropriate available expertise, or conversely, because the agents with the best expertise may not know about appropriate tasks for them. Similarly, the incomplete knowledge of agents may prevent consistent and complete role assignment because there may be no one agent that has a

global knowledge of all the roles or subtasks that need to be assigned. In a single-agent problem solving situation, this issue does not arise. The agent knows how it has decomposed a task into subtasks, and it knows exactly which subtasks it has to do--all of them.

Once tasks or roles have been assigned, distributed problem solvers face severe difficulties in *coordinating task execution*. Like single-agent tasks or subtasks, group tasks may not be independent. Temporal or logical dependencies may exist. For example, if the group problem is to build a new chip, the designer's role must be completed prior to the initiation of the manufacturer's role. In addition, tasks that are not logically connected may interact through shared resources. For example, if two blocks-world agents are each to build towers, one agent's plan will negatively interact with another's if both intend to use the same block (Davis, 1981). The interaction is negative because the first agent is satisfying its task, but at the cost of preventing the second agent from doing the same. In contrast, the plans might interact positively, say, if one agent's plan entails using (hence picking up) a block that currently lies on top of the block another agent intends to use. The interaction is positive in the sense that the first agent is not only satisfying its task, it is also helping the second agent satisfy its task.

Single-agent problem solvers have difficulties in handling nonindependent tasks or subgoals (Sussman, 1975), but these difficulties multiply for distributed problem solvers, because of limited knowledge. If two agents have only local knowledge--e.g., if they know only the local environment and only their own tasks and intentions--they will not be able to prevent negative interactions between goals or roles. If the chip designer does not know about the chip manufacturer, there is no basis for coordinating their subtasks; if one blocks-world agent doesn't know the intentions of another, there is no basis for ensuring that their projected uses of resources will not conflict. Similarly, without some knowledge of others' tasks and intentions, positive interactions, the essence of effective group problem solving, cannot be encouraged.

In summary, the main challenge in distributed problem solving is to make the solutions a distributed agent produces not only locally acceptable, achieving the assigned tasks, but also interfaced correctly

with the actions of other agents solving dependent tasks. The solutions must not only be reasonable with respect to the local task, they must be *globally coherent*, and this global coherence must be achieved by *local computation alone*. Global coherence is less difficult to achieve for a single-agent problem solver, simply because its computation and knowledge are themselves as global as the task requires.

III. STRATEGIES FOR COOPERATION

How can global coherence be achieved in distributed problem-solving groups, in the face of limited knowledge and the requirement that all computation be local? Intuition says that it can be achieved, since there are cases where groups act synergistically, solving problems *better* than any individual could. Broadly, the key for coherent distributed problem solving lies in the fact that while distributed agents have greater difficulties in solving a given task, they have potentially more options as well. A single-agent problem solver must gather all information itself; distributed agents work singly as well, but they may also ask others to help. A single-agent problem solver must perform all planning itself; a distributed agent may plan or act, but it may also request others to do so, resulting in speed through parallelism. In short, much of the power of distributed problem solving comes through cooperation and communication.

We have come to believe that there are no general algorithms to dictate optimum cooperation. Methods that yield good distributed performance under one set of conditions fail under others. Although communication between agents provides the basis for effective cooperative problem solving, it is just another problem-solving tool that may be used either poorly or effectively. If the tool is used poorly, then group problem-solving performance may be worse than individual problem-solving performance. It requires considerable expertise to use communication effectively. This expertise seems to take the form of a broad range of heuristic rules. We refer to such expertise collectively as *cooperative strategies*. Our main theoretical and empirical goals have been to understand such strategies. In a theoretical vein, we have attempted to analyze the components of cooperative strategies and the range of alternative strategies that may be adopted. More empirically, we have attempted to determine the performance characteristics of such strategies and the conditions under which each will perform poorly or effectively. We have classified these heuristic cooperation strategies under two headings: *organizational*

policies and *information distribution policies*. In the following sections, we briefly discuss some theoretical aspects of cooperation. Subsequently, we present some empirical tests of several specific policies we have implemented in a distributed problem solver for air-traffic control. We finish by describing some further implementations in RPV coordination.

ORGANIZATIONAL POLICIES

Organizational policies dictate how a larger task should be decomposed into smaller (sub)tasks which can be assigned to individual agents. Typically, a given organizational policy assigns specific roles to each of the agents in a group. Such a policy is useful if for some tasks the resulting division of labor enables agents to work independently. For example, the corporate hierarchy is an organizational policy that is particularly effective if the corporate task can be decomposed in such a way that an agent at one level can work independently of others at that level, reporting results only to its immediate superior, who takes care of any necessary interfacing.

Organizational policies not only define task decomposition, but they also prescribe communication paths among agents. They turn a random collection of agents into a network that is fixed, at least for a given task. In the corporate hierarchy, again, the arcs between agents usually indicate which pairs are permitted to talk to one another and, in addition, they determine the nature of the messages that are allowed. Such communication restrictions are beneficial if they encourage only those agents who should communicate to do so--in particular, agents who have dependent tasks or who may share resources. In general, organizational policies strongly direct and constrain the behavior of distributed agents. If those constraints are appropriate to the task at hand, then the organization is effective; otherwise, its performance may be suboptimal.

Agents must know not only which policy is appropriate to the current circumstances, but also the *techniques by which a group can implement the chosen policy in a distributed fashion*. How is the assignment of roles specified by the policy made to agents? How is the agent that is most appropriate for a given task found?

Briefly, any distributed method of implementing an organizational policy must answer a variety of questions, including:

- Are agents externally directed or data-directed (Lesser, 1981)? That is, does an agent arrive at its roles by being told them, or is information relayed, allowing the agent to assign the roles itself?
- When an agent is requested by another agent to conform to a role or to take on another subtask, does the first agent have the right to negotiate?
- How does an agent weigh the value of competing tasks?

Smith (1978) has proposed the contract net as a formalism for implementing certain organizational policies in a distributed fashion. Other possible distributed policies can be derived from the blackboard scheduling techniques of HEARSAY-III (Erman, 1981) and AGE (Nii, 1979), and from hierarchical control structures used in production systems (see, for example, the goal-oriented control in OPS5 (Forgy, 1981)). In Sec. VI, we discuss a somewhat different organizational policy implemented using our framework.

INFORMATION-DISTRIBUTION POLICIES

An information-distribution policy addresses the nature of communication between cooperating agents. Decisions about how agents communicate with each other are, first of all, constrained by the choice of organizational policy, since that policy decides the network of permissible communicators. However, within these constraints, a great number of lower-level decisions must be made about how and when communications should occur:

- *Broadcast or selective communication.* Are agents discriminating about whom they talk to? If so, what criteria are used to select recipients?

- *Unsolicited or on-demand communication.* Assuming an agent knows whom it wants to communicate with, does it do so only if information is requested, or does it infer the informational needs of other agents and transmit data accordingly? What form of information (data, constraints, commands, goals) should it send?
- *Acknowledged or unacknowledged communication.* Does an agent indicate that it has received information?
- *Single-transmission or repeated-transmission communication.* Is a piece of information sent only once, or can it be repeated? How frequently? Lesser (1981) refers to a repeated-transmission policy as murmuring.

Poor decisions at this level result, at best, in the highly inefficient use of limited-bandwidth channels. At worst, such choices endanger global coherence by preventing agents whose tasks may interact from talking to one another. The goal of information-distribution policies is to minimize these possibilities. As with organizational policies, the choice of communication policies depends on current conditions. These include the bandwidth of the communication channel, the reliability of the channel, the load of the channel, the maximum acceptable information turnaround time, and the relative cost (and time) of computation versus communication. Effective communications management also requires accurate modeling of other agents' knowledge.

IV. A FRAMEWORK FOR DISTRIBUTED PROBLEM SOLVING

The previous sections gave an analysis of distributed problem-solving situations, as compared with single-agent problem-solving environments, and informally described the special competences that multiple-agent problem solvers must possess. In the following sections, we attempt to develop a computational theory of these cooperative capabilities. Our goal here is to explain how organizational policies and information-distribution policies can be computed, and then to describe specific implementations of such policies.

REASONING ABOUT TASKS

What sorts of computations are involved when groups successfully employ a cooperative strategy? Abstractly, such successful cooperative actions require each agent to make decisions about:

- Which of planning, execution, evaluation, etc., it should do.
- When it should do them.
- How it should do them.
- Whom it should talk to.
- When it should communicate.
- What it should say.

In short, each agent needs to make decisions about its preference among tasks, their timing relative to one another, and their content. It needs to *reason about possible tasks*.

Reasoning about possible tasks plays a much less significant role in single-agent problem-solving situations than in distributed problem solving. Many theories of individual problem solving (Fahlman, 1974; Sacerdoti, 1974) suggest that a problem solver's activities are decomposed into separate, strictly ordered phases of information gathering, planning, and execution. Because this task ordering is so trivial, there is little need to reason explicitly about it. Almost all reasoning goes on *within the planning task*. Unless the problem solver

must maintain multiple lines of reasoning or deal with time-varying data, there is little or no meta-level reasoning *about planning*--whether it should be done, how it should be done in relation to a variety of other tasks that comprise the problem-solving process. However, the competences we described earlier indicate that a simple, fixed ordering of tasks is not possible for multiple-agent problem solvers. The ordering of tasks is not simple, because there are many tasks to manage and they are frequently nonindependent (e.g., the agent must receive and send communications as well as plan and execute). The ordering of tasks cannot be fixed but must be *dynamically changed*, because agents will be accruing information about the environment and the activities of the other agents and will need to alter their task preferences on the basis of this new information.

THE FRAMEWORK: TENETS

To achieve a computational understanding of distributed problem solving, we therefore need a vocabulary that enables us to formally represent task entities, and to formally express rules that reason about the represented tasks. We have developed a framework which meets these needs. Briefly, the main tenets of the framework are:

1. Each agent has several distinct kinds of *generic tasks*, such as information gathering (sensing and input-communication), information distribution (output-communication), plan generation, plan evaluation, plan fixing, and plan execution.
2. Each kind of generic task invocation (or *task instance*) is a process: It can be suspended and resumed, hence tasks can be interwoven without losing *continuity*.
3. Each agent has a knowledge base that represents its beliefs about other agents and their intentions, as well as information about the static environment and its own intentions.
4. Within an individual agent, the knowledge base is shared by all task instances, like a HEARSAY blackboard (Erman, 1981). Any change in the knowledge base made by a task (e.g., information gathering) while another task is suspended (e.g., planning) will be visible to the latter when it resumes. Thus tasks such

as planning exhibit *currency* as well as continuity; they do not base computations on an outdated world-model.

5. Task instances are both data-driven and event-driven. Instances of generic tasks are triggered in two ways: by sets of well-defined knowledge-base states, or by well-defined events which result in changes to the knowledge base. Tasks that are created do not immediately get executed but are enabled and may compete for processing resources.
6. Each enabled task has a limited amount of self-knowledge, including explicit intentions and validity conditions. This information can be used to determine if a task is justified in continuing as conditions change. Thus tasks will exhibit *relevance*.
7. Enabled tasks are not invoked in a fixed order, as in single-agent problem solvers. Rather, the agent acts as a scheduler, reasoning about the ordering of task instances. More specifically, the agent uses a set of *heuristic reasoning rules* to prioritize processes representing enabled tasks.
8. A task selected by the agent for execution is not necessarily allowed to run to completion. It is given an upper limit of processing resources (time). The extent of this limit is also controlled by the agent.
9. During the execution of a task or process, (a) the task may complete, in which case it is eliminated from the set competing for resources; (b) new tasks may be created because of knowledge-base changes or events effected by the running task; (c) the changes may cause existing tasks to lose their justification.
10. After a task has consumed its allocated supply of resources (i.e., time), the agent reorders the priority of enabled tasks and selects a new one to run, in light of the conditions in the altered knowledge base. It also eliminates unjustified tasks (if the tasks have not eliminated themselves).
11. This procedure iterates until there are no more enabled tasks worth running.

Generally, then, we view the agent in a group problem-solving situation as a kind of knowledge-based operating system. The view is not a model of an agent in a specific distributed domain, but rather represents a theoretical framework for describing distributed agents or a set of guidelines for constructing a specific model. The framework is similar to that used in blackboard systems (Erman, 1981) but is more dynamic, in that tasks can be interrupted at any point. Adhering to the framework, the user still needs to provide several sorts of domain-specific expertise, including the procedures that comprise each generic task, the triggering conditions under which a task instance is to be created, the validity conditions under which it is permitted to continue, and the heuristic rules that order the priority of enabled tasks in light of the current state of knowledge.

THE FRAMEWORK: IMPLEMENTATION

To facilitate the development of our specific distributed problem solvers, we have implemented the framework in a simple task language. The task language is a set of INTERLISP functions that provide the user with a convenient vocabulary stating the required domain-specific expertise. Once stated, the task language takes care of all the specifics of task management. It insures that an appropriate task instance is enabled whenever the triggering conditions of a user-defined generic task are met. The task language also takes care of the low-level implementation of tasks as resumable coroutines, and it guarantees that these processes suspend after consuming the appropriate amount of time. Finally, it handles the details of scheduling the next task to run; the user needs only to state the reasoning rules of the scheduler that its application requires.[1] By attending to the details of task creation and management, the task language frees the user to focus on the theoretically more interesting issues of designing (and debugging) rules that achieve the appropriate interweaving of tasks.

---

[1] For more details on the capabilities of the task language, see McArthur et al., 1982.

Our task language can be compared with the several specialized artificial intelligence languages built on top of LISP. Such languages were developed to provide a special set of primitives for building programs in a limited domain. For example, EMYCIN (van Melle, 1979) facilitates the development of diagnostic systems like MYCIN (Shortliffe, 1976) and PUFF (Kunz et al., 1978)--although it may not help in constructing fundamentally different systems--by providing "expert-systems" concepts as primitives. To the extent that the primitives provided actually suit the intended domain of application, they simplify the programmer's design and implementation problems. In the next section we test out the primitives of our task language by employing the language to implement several distributed air-traffic-control problem solvers. Concrete examples will be given of how agents, tasks, and rules for reasoning about tasks are represented.

V. AIR-TRAFFIC CONTROL AS A DISTRIBUTED PROBLEM

Problem solving in air-traffic control (ATC) may be distributed in several ways. In Steeb et al. (1981), we discuss a variety of architectures of distribution. Our present systems are all *object-centered*, with an agent associated with each aircraft. That is, each aircraft has its own onboard planning, control, and communication systems. In our ATC task, aircraft enter a rectangular (14 x 23 mile) airspace at any time, either at one of 10 infixes (entry points) on the borders of the airspace or from one of two airports. Figure 1, taken from our ATC task simulation, shows the airspace in a relatively congested state. The main goal of an agent is to navigate its associated aircraft through the airspace to an assigned destination--

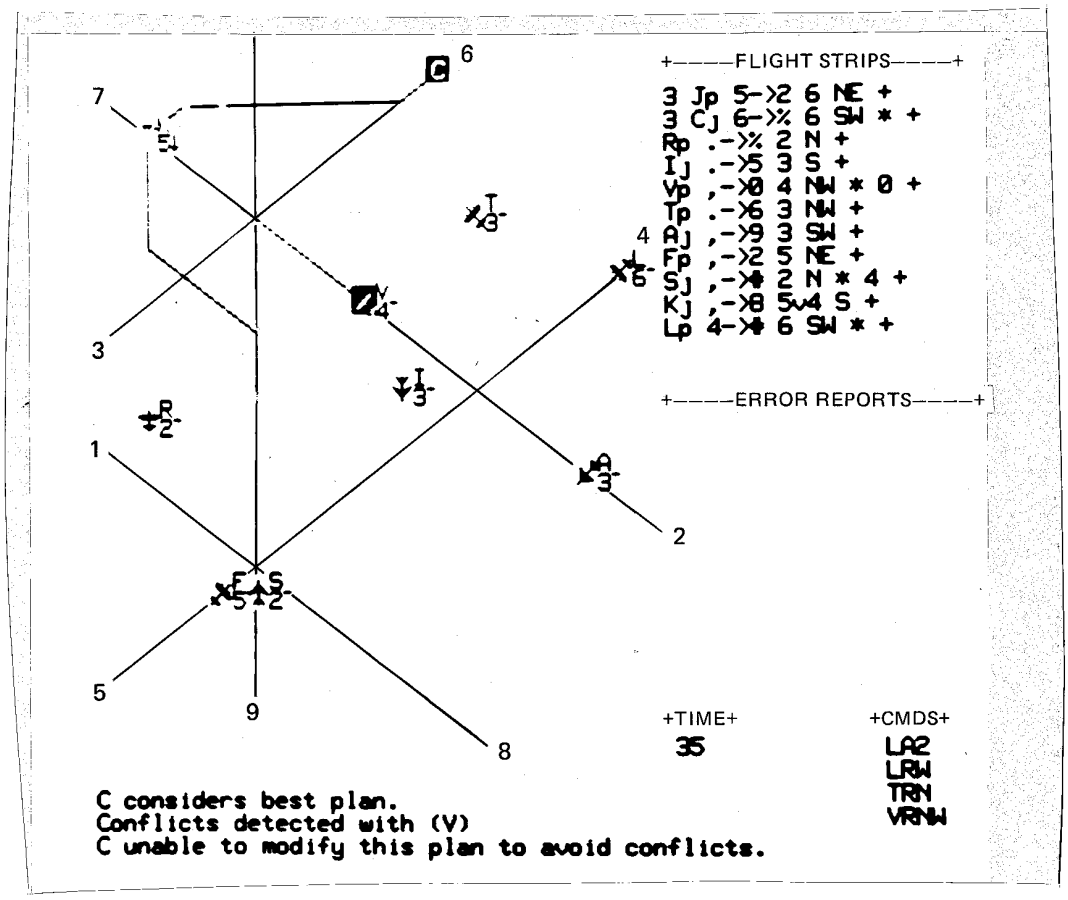

Fig. 1 -- Air-traffic control task simulation

either a boundary outfix or an airport. Each aircraft has only a limited sensory horizon, hence its knowledge of the world is never complete, and it must continually gather information as it moves through the airspace. Information may be accumulated either by sensing or by communication. Agents are allowed to communicate over a limited-bandwidth channel to other aircraft for purposes of exchanging information and instructions.

Distributed ATC is a group problem not only because agents may help one another gather information, but also because the goals of one agent may interact with those of another. Goal interactions come in the form of shared interaircraft conflicts. A conflict among two or more agents arises when according to their current plans, the agents will violate minimum separation requirements at some point in the future. When shared conflicts arise, agents must negotiate to solve them. In a crowded airspace, such goal conflicts can become particularly complex and may involve several aircraft, necessitating a high degree of group cooperation.

In terms of the vocabulary developed in Sec. II, the detection and resolution of conflicts are the main distributed problem-solving *tasks*. These tasks may be decomposed into several *subtasks*, or distinct *roles*. Agents may gather information about a shared conflict, evaluate or interpret the information, develop a plan to avoid a projected conflict, or execute such a plan. Agents may be more or less *appropriate* for such roles, depending on their current processing load (Are they currently involved in helping resolve other conflicts?), their state of knowledge (Do they know a lot about the intentions of other agents in the conflict?), and their spatial constraints (Can they locate many nearby aircraft through sensing and do they have much excess fuel?).

The issue of *optimal task assignment* arises because a group of aircraft may fail to assign the most appropriate agent to each role in a conflict task if some of the aircraft do not know about a shared conflict. In addition, care must be taken to assign a complete and consistent set of roles. Some role inconsistencies can be fatal. For example, two agents would be adopting inconsistent roles if one decided to move left to avoid a head-on collision with the second, while the

second decided to move in the same direction. Severe *task coordination problems* may also arise in distributed ATC. The action of moving to avoid one conflict may create or worsen other conflicts (negative task interactions) or it may lessen other conflicts (positive task interactions). Both forms of interaction are caused by the fact that while agents may be dealing with different conflict tasks, they are nevertheless exploiting shared, limited spatial resources.

VI. FOUR DISTRIBUTED PROBLEM SOLVERS FOR AIR-TRAFFIC CONTROL

We outline in this section the implementation of four distinct ATC systems, concentrating particularly on the organizational and information-distribution policies embedded in each. All four systems are implemented in our framework for constructing distributed agents. This in turn is implemented in INTERLISP-D, running on Xerox 1100 computers. Before discussing the implementation of cooperative strategies, we turn first to a general description of the the tasks that must be reasoned about by a distributed ATC (DATC) agent.

TASKS IN AIR-TRAFFIC CONTROL

To define our system within the framework of the task language, we must identify the tasks comprising each agent and specify the expertise associated with each task. The top-level generic tasks of each DATC agent currently include:

- Sensing (gathering information about positions and types of other aircraft).
- Input-communication (gathering information about routes, plans, and requests of other aircraft).
- Output-communication (distributing information about the agent's routes, plans, and requests to others).
- Initial plan generation (computing a reasonable path through the airspace to one's outfix).
- Plan evaluation (finding conflicts between the agent's plan and the plans it believes others are following; reviewing new information for consistency with beliefs about others' plans).
- Plan fixing (using existing plans and evaluations to create new plans that avoid conflicts with others).
- Plan execution (performing the time-tagged actions called out in the plan).

Defining ATC Generic Tasks and Conditions of Invocation

A major part of defining a generic task is stipulating the conditions under which an instance of that task should be created. Consider plan evaluation: We want to define the ATC agent so that an evaluation task is created, when (a) the agent has a plan and, via some information-gathering task, learns the plan of some other aircraft, (b) the agent changes its own plan, or (c) the agent believes it knows the plan of another aircraft and senses a new position for that aircraft that may not be consistent with what the believed plan predicts. In the first two cases, the kind of evaluation needed is "conflict detection"; in the third, it is "consistency checking." Using the task language, the "conflict detection" case is implemented as follows:

```
(1)  (CREATE-SUBTASK-TYPE 'Evaluation 'Scheduler)
(2)  (CREATE-SUBTASK-TYPE 'DetectConflict)
(3)  (SET-TASK-FUNCALL 'DetectConflict
                        '(COMPUTE-CONFLICTS Aircraft OtherAircraft))
(4)  (DEFINE-TASK-TRIGGER 'DetectConflict 'Evaluation
         '(SET-AIRCRAFT-PLAN OtherAircraft y)
         '(Check new plan of OtherAircraft for conflicts against yours)
         '(AND (AIRCRAFT-PLAN OtherAircraft)
                 (EQUAL y (AIRCRAFT-PLAN OtherAircraft))))
```

Line (1) establishes the generic task of plan evaluation. *Evaluation* can be thought of as a class object in the SMALLTALK sense (Goldberg and Kay, 1976). Instances of *Evaluation* represent specific plan evaluation tasks that might be created. The second argument in line (1) says that when a plan evaluation task is created it is to be a subtask of the current instance of *Scheduler*, the top-level generic task of the ATC agent. Only one instance of *Scheduler* is ever created for each agent, and its role is to select the next enabled top-level task to execute (e.g., sensing, planning, input-communication, etc.).

Line (2) establishes a generic subtask of plan evaluation. When triggering conditions of *DetectConflict* are met and an instance of it is created, the instance becomes a subtask of the current *Evaluation* task of the agent. Thus while the agent's *Scheduler* task chooses from among

enabled tasks that are instances of generics such as *Evaluation* and *Sensing*, an *Evaluation* instance itself is a scheduler that chooses from among instances of *CheckConsistency* and *DetectConflict*.

Line (3) associates a function call with *DetectConflict*. When an instance of a generic task becomes enabled, it may be selected to execute by the *Evaluation* task. If the task has previously executed and suspended, *Evaluation* knows where to resume; if this is the first time the task has been allocated processing resources, *Evaluation* needs to have a way of initiating the task. It does this by evaluating the function call. Line (3) presupposes that COMPUTE-CONFLICTS has been defined by the user and encodes the appropriate expertise.

Line (4) stipulates the conditions under which task instances of *DetectConflict* will be created and will become a subtask of *Evaluation*. The interpretation of DEFINE-TASK-TRIGGER is:

```
(DEFINE-TASK-TRIGGER
  "create an instance of this type of generic task"
  "let the scheduler of the new instance be the current instance
   of this generic"
  "create the instance whenever a form of this type is evaluated"
  "let this be the intention of the created task instance"
  "this form must always be true for the instance to be justified")
```

Thus line (4), for example, says, "Any time you believe you know some other aircraft's plan, it is reasonable to create a *DetectConflict* task, as a subtask of the current *Evaluation* task, to see if your current plan conflicts with its new one. This task is justified as long as you still believe you know the aircraft's plan and it is the new one."

Defining Reasoning Rules that Interweave ATC Task Instances

Declarations such as line (4) in the conflict-detection example show how task creation is data-driven, how tasks ensure that they are relevant as conditions change, and how tasks may be suspended and resumed. But to be able to reason about how to intelligently interweave tasks such as plan evaluation, information gathering, etc., permitting the DATC agent to perform intelligently, we still need to define heuristic rules that will reason about the priority of enabled tasks. Two reasoning rules currently used are:

```
(1)  (DEFINE-SCHEDULING-RULE 'Scheduler
       (if (TASK-TYPE Subproc)='PlanFixing
           and (SUBTASK-OF-TYPE Process 'Evaluation)
         then (SET-TASK-PRIORITY Subproc 0)))

(2)  (DEFINE-SCHEDULING-RULE 'Scheduler
       (if (TASK-TYPE Subproc)='SendReplanRequest
           and (SUBTASK-OF-TYPE Process 'PlanFixing)
           and (GREATERP (TASK-TOTAL-TIME
                                (SUBTASK-OF-TYPE Process
                                                 'PlanFixing))
                                          5000)
           and (NOT (IN-IMMINENT-DANGER Aircraft))
         then (SET-TASK-PRIORITY (SUBTASK-OF-TYPE Process
                                                  'PlanFixing)
                                           0)
              (SET-TASK-PRIORITY Subproc 200)))
```

Rule (1) defines a choice of the DATC agent's scheduler; thus it helps the agent decide which of the enabled top-level tasks to execute next. The rule says that if *PlanFixing* is enabled (because an aircraft's plan has a conflict in it), then it is a good idea not to allocate further resources to this task if there is some evidence that the conflict status of the plan should be reevaluated. The rationale is that the *Evaluation* task may have been enabled by receipt of a new plan for the aircraft causing the conflict, and this plan may avoid the conflict.

Rule (2) also defines a top-level reasoning process for the DATC agent. Details aside, its role is to decide when a given agent (aircraft) has tried "hard enough" to solve a conflict shared with another aircraft. Note that "hard enough" has a natural definition in terms of the processing resources (time) that have already been devoted to attempts at *PlanFixing*. If this criterion is met, the agent will use its other option in solving a shared conflict: It will ask the other conflictee to try to resolve it (by invoking the *SendReplanRequest* task) instead of expending more effort to try to resolve the conflict itself.

Rules such as (1) and (2) are the key to the DATC agent's ability to interweave its several enabled tasks in a way that is sensitive to changing conditions. Many of the rules the DATC problem solver currently employs are devoted to ordering tasks that are purely "internal" to the agent. These tasks, which include sensing,

evaluation, plan fixing, and plan execution, often must be interwoven because of the existence of external, unpredictable, agents. The tasks are affected by and do not directly involve those agents. On the other hand, rules like (2) reason about tasks that involve interaction (communication) with others, in the service of either one's own goal or others' goals. The resulting group interactions tend to be much like the dynamics of constraint propagation (Stefik, 1981), or least-commitment planning (Sacerdoti, 1977). An agent pursues a planning option until it is unable to satisfy all constraints or until its alloted time runs out. It then suspends its planning, requests another agent to replan, and sends its own partial plan as a constraint. The process continues without backtracking. In the following section, we discuss the implementation of such strategies in four different ATC problem solvers.

ORGANIZATIONAL POLICIES IN ATC

Four organizational policies for dictating task decomposition and role assignment are discussed below. The organizational policy embedded in three of the four systems may be characterized as *task centralization*; the fourth system adheres to a policy of *task sharing*. Under *task centralization*, the agents involved in any given conflict task will choose one of their number to play most of the roles. In effect, one agent will perform the evaluation role (do all the evaluation of the potential conflicts between aircraft), the plan-fixing role (attempt to devise a plan-fix to dissolve the entire conflict), and the actor role (act on the new plan). The selected agent is required to modify only its plan to resolve the conflict; thus the remaining agents perform no planning or actions. Instead, having agreed on the choice of a replanner, they adopt passive information-distribution roles, merely sending their intentions (plan) to the selected agent. As mentioned earlier, if the selected agent is unable to resolve the entire conflict, he requests another agent to replan. This process continues until all conflicts are resolved or a solution cannot be found. The policy of task centralization, whatever its shortcomings, is worth considering, because it has many of the advantages of the centralized, single-agent problem solving that it is meant to mimic. Specifically, by

centralizing most task roles in a single agent, the group has to worry less about negative task interactions such as the threat of two aircraft acting in an inconsistent fashion, noted above.

Although three of our four systems embed a task-centralization policy, they differ in how they measure and choose the agent that is most appropriate for the several centralized roles. Also, these schemes represent forms of distributed problem solving in spite of being termed centralized, because many conflicts may be resolved simultaneously by different aircraft over the airspace. The term *centralized* applies only within a given instance of a conflict.

Selection by Shared Convention. In selection by shared convention, each aircraft uses only directly sensed information about the other aircraft (position, heading, and speed) to decide which should plan and which should transmit its current route. The aircraft silently use a common set of conventions for this decision, minimizing communications. Figure 2 shows a prototypical sequence of tasks, including communication tasks, between two aircraft, A and B, under this policy. Each entry in the time line for an aircraft represents the execution of a task instance, using task triggers and scheduler reasoning rules such as the ones presented above.

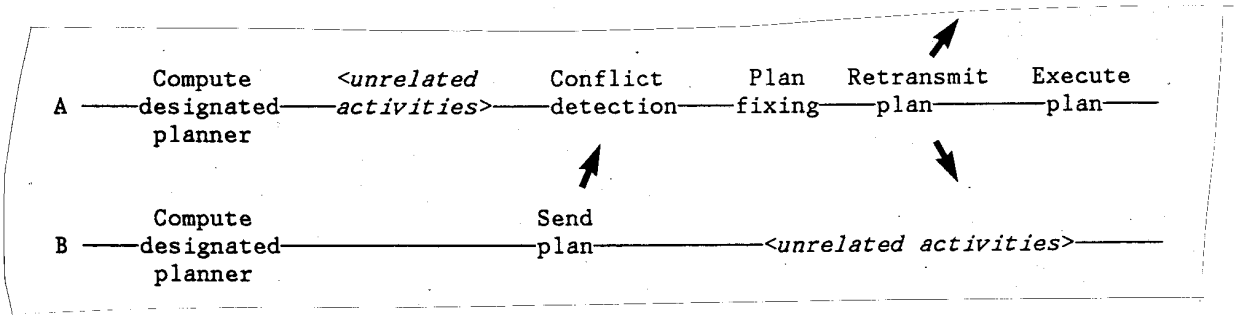

Fig. 2 -- Prototypical task sequence under the shared-convention policy (time lines are for tasks executed by aircraft A and B; solid lines indicate communications)

Because of the limited criteria used, the aircraft selected as the replanner is not likely to be the most appropriate. This version mainly serves as a benchmark against which to judge the utility of more intelligent methods of selection, which are also more costly in terms of computation and communication.

Selection of the Least Spatially Constrained Agent. With this selection method, each aircraft in a potential conflict computes and transmits its *role factor* to the other aircraft. The role factor is an estimate of the appropriateness of an aircraft for the planning role; it results from the constraints under which an aircraft is operating. It is an aggregation of such considerations as the number of other nearby aircraft, fuel remaining, distance from destination, and message load. Figure 3 shows the standard sequence of tasks and communications under this policy.

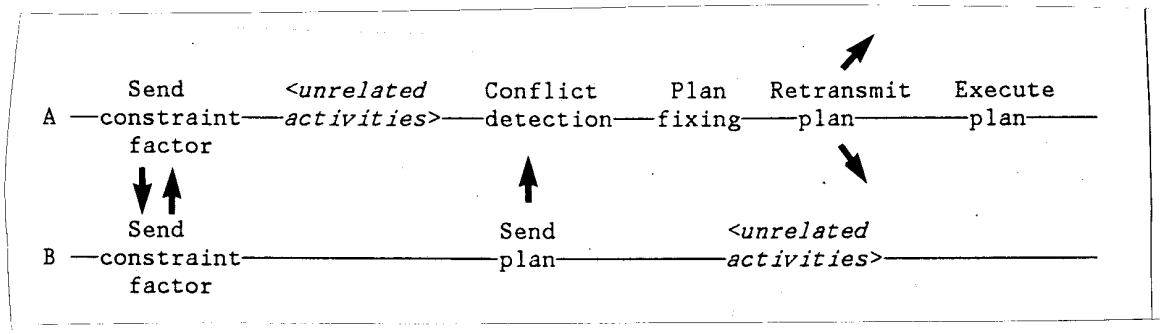

Fig. 3 -- Prototypical task sequence under the least spatially constrained policy

This method of selection maintains that the most appropriate agent is the one with the most degrees of freedom for modifying its plan. It is a more complex process than the *shared convention* and should result in more effective replanner choices, although at some additional cost in initial communications.

Selection of the Most Knowledgeable, Least Committed Agent. As in the above selection scheme, aircraft share *role factors*, but here they are computed differently. This method of selection maintains that the best replanning agent is the one that knows the most about other agents' intentions, because, in replanning, a well-informed agent can explicitly take account of possible interactions between its intentions and those of other agents. More globally coherent plan-fixes should therefore result. In addition, this method says that agents whose intentions are known by others should not replan. If such an agent does modify its plan, it will have violated the expectations of cooperating agents, making their knowledge incorrect and in turn making cooperation

difficult. Thus, this policy implements a common adage of cooperation: Don't do the unexpected.

In spite of their simplicity, task-centralization policies are often ineffective. Although the agent selected to perform the centralized roles may be the best *overall*, that agent is rarely the best for *each* of the centralized roles. For example, we still might want to assign the actor role to the agent in a conflict set which is least constrained in the sense defined above. However, that agent might not be the best in the set for fixing its plan--for making a modification to the plan and evaluating the implications of such a change. Presumably the best agent for this role is the (possibly distinct) member of the conflict set that knows most about the environment and the intentions of aircraft near the one whose plan is to be fixed. This aircraft is in the best position to determine whether any changed plan is not only locally reasonable, solving the conflict, but also globally reasonable, not creating new conflicts with other aircraft.

Task Sharing

The *task-sharing* policy attempts to avoid such problems by evaluating agents' qualifications with respect to each of the roles associated with a conflict. Whereas in centralized policies a single negotiation determines an overall replanner, in the task-sharing policy two rounds of negotiation are necessary, one to determine the plan-fixer and one to determine the actor. Figure 4 presents a prototypical sequence of tasks and communications showing how such a policy is implemented in a distributed fashion.

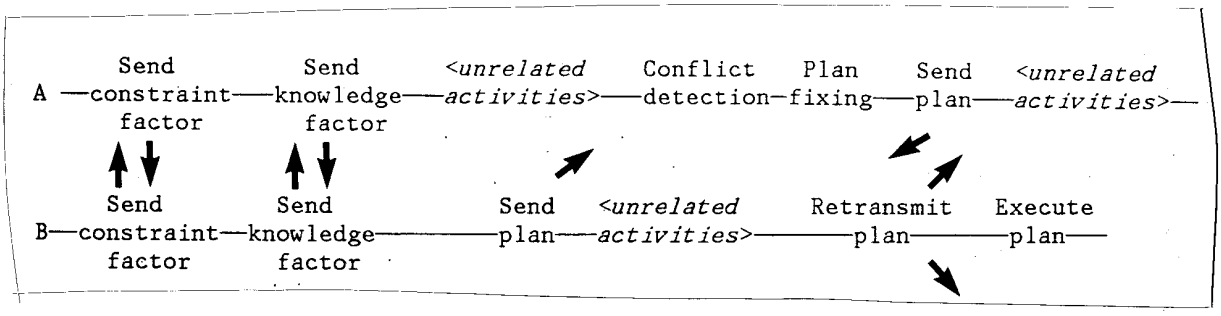

Fig. 4 -- Prototypical task sequence under task-sharing policy

The performance of groups working under a task-sharing policy is potentially superior to that of groups working under a task-centralization policy, because in the former the group attempts to optimize on each role. However, in practice this policy has several possible drawbacks. It is communication-intensive and may be inappropriate when communication channels are unreliable or costly. Moreover, it risks potential negative interactions, because several agents have to coordinate intimately to achieve a solution.

INFORMATION-DISTRIBUTION POLICIES IN ATC

Much of the information-distribution behavior in the four systems is set by the choice of organizational policy (who to contact, what to send, when to send it). We assume in all cases that information should be sent to other aircraft selectively (no broadcasting), without waiting for a request, without expecting an acknowledgment, and without repeating the information a second time. These choices are reasonable, since we assume in all systems that communication is error-free. When we add noise to the communication channel, we envision adopting a policy that injects some needed redundancy or safety into communication, for example, a policy that includes murmuring (Lesser, 1981). We also assume a constant effective communication bandwidth for all four systems. Each aircraft is allowed to send a maximum of 5 messages per 15 seconds of time.

VII. EXPERIMENTAL STUDIES IN AIR-TRAFFIC CONTROL

We conducted a series of rudimentary experimental studies on the four policies outlined above. We focus here on results pertaining to the three task-centralization policies, since our studies on the task-sharing policy were performed later and were limited in scope. The task-centralization variants were tested on eight distributed scenarios. Each scenario stipulated (1) how many aircraft would enter the airspace in the session, (2) when and where they would enter, and (3) where they would exit. This control over the parameters of distributed problem-solving situations allowed us to isolate situation features that uncovered the strengths and weaknesses in performance of our policies. We varied the scenarios considerably in task density, time stress, and task difficulty. The primary factor affecting these conditions was the number of aircraft in simultaneous conflict.

We examined three performance indices when comparing the systems: communication load, processing time, and task effectiveness. Task effectiveness was indicated by two distinct factors: separation errors (more important) and fuel usage (less important). A summary of the main results is given in Table 1.

We found that the *shared-convention* policy, relying on essentially arbitrary assignment of planning responsibility, performed well only in low-complexity, low-difficulty tasks. It minimized communications and response times compared with the other policies but it quickly foundered in three- and four-body conflicts.

Of the three task-centralization strategies, the *least constrained* policy performed best. It did particularly well on high-complexity, high-difficulty tasks. In such cases, the planning aircraft tended to be located at the edge of the fray, able to find more viable solutions than the aircraft in the interior. The policy is time- and communication-intensive, however, largely because of the high number of messages needed to cooperatively determine the replanner and to maintain consistency after replanning. In any of the three task-centralization systems, when a replanner is successful it must send *data retransmission*

Table 1

PERFORMANCE MEASURES OF THREE ORGANIZATIONAL POLICIES
(statistics averaged across 8 scenarios)

Item	Shared Convention	Least Constrained	Most Knowledgeable
Communication load[a]	10.9	28.6	28.2
Processing time[b]	1265	1726	1651
Separation errors[c]	4.3	1.4	2.3
Fuel usage[d]	96	108	101

[a]Mean messages sent per aircraft while flying from infix to outfix.

[b]Mean Xerox 1100 cpu seconds per aircraft while flying.

[c]Mean number of near misses or collisions for all aircraft in a scenario.

[d]Mean number of fuel units used for all aircraft.

messages to all aircraft to which it had previously sent its intentions. The number of data retransmissions was especially high under the *least constrained* policy.

The *most knowledgeable* policy was intermediate in performance. It performed best in tasks of low complexity and high difficulty, that is, tasks with primarily two- and three-body interactions and few potential solutions. In complex multiaircraft situations, if the wrong aircraft was chosen for planning, the result was often catastrophic, because the aircraft that then received replan requests tended to have little knowledge of the routes of other aircraft. By design of the policy, this knowledge was typically concentrated in the initially selected planner. That planner normally continued to be the most knowledgeable in later interactions.

When successful, the *most knowledgeable* policy's performance was in some ways better than that of the *least constrained* policy. In particular, when an agent found a solution to a local conflict task under the *most knowledgeable* policy, that solution was likely to be more globally coherent than solutions found under other policies, since the

replanning agent was selected partially because of its wide knowledge of the plans of the other aircraft. This knowledge allowed it to more effectively replan without incurring new conflicts. In addition, a successful replanning agent under a *most knowledgeable* policy generally needed to issue fewer data retransmission messages than under the other policies, since it was selected partially because its intentions were known to fewer others (i.e., it was the least committed agent). We had initially anticipated that minimizing data retransmissions would be very important for guaranteeing globally coherent performance. We envisioned situations where one retransmission would cause the receiving agent to reevaluate, possibly finding new conflicts, causing more replanning, further data retransmissions, and so on, in a vicious propagation of changes. This did not happen as often as we had expected under the *least constrained* policy, although a few instances were observed.

Another erroneous expectation was that there would be a wide variation in processing times among the aircraft under the *most knowledgeable* policy. This policy should tend to bias replanning in favor of a few agents. If an agent is the replanner once, it gains new knowledge of others' plans, making it an even better choice as replanner for later conflict tasks. We anticipated that this concentration would skew the processing times, compared to a more uniform distribution of responsibilities under the other policies. This would have been a disadvantage in a truly distributed system, as some agents would be quiescent much of the time. The expected variation in times did not evidence itself, however, except in the relatively easy scenarios.

While limited in scope, the data collected from our fourth policy, *task sharing*, indicated some interesting trends. This policy, a composite of the best of the *least constrained* and *most knowledgeable* policies, had the advantage of choosing one agent to act, and another with more knowledge of the situation to compute the first agent's plan. This policy was often effective in situations where subtasks are easily separable and an explicit selection of the agent with the best available expertise could be made for each subtask individually, rather than for the conflict task as a whole.

VIII. REMOTELY PILOTED VEHICLE FLEET CONTROL

We next moved to a richer and more difficult domain: surveillance RPV fleet control. Coordination of groups of military RPVs is a much more demanding application of distributed problem solving than air-traffic control, because of unreliable communications, distinct roles for each aircraft, needs for coordinated actions, and frequent attrition from the hostile environment. In this section, we describe a new RPV fleet-control implementation embodying many of these problem aspects. We discuss some of our preliminary findings relating to role assignment, data fusion, communication management, and cooperative planning.

TASKS IN RPV FLEET COORDINATION

One of the principal goals of RPV development today is to realize greater vehicle autonomy. Current RPV technology--represented by Israel's Mastiff (Smith, 1983) and the Lockheed Acquila (Hyman, 1981)--relies on the use of close in-the-loop control by skilled remote human operators. Each operator controls a single RPV through a continuous and vulnerable communication link. Operations during radio silence or jamming are usually confined to a few preprogrammed actions (such as spiraling up to regain contact, continuing on the same path, and self-destructing). The few attempts at multiple RPV control, such as IBM's large-scale experimental system (Gray et al., 1982), have relied entirely on a vulnerable centralized airborne or ground center to perform all control operations. We hope to extend this technology by developing techniques for onboard autonomous or semi-autonomous planning and control. Such capabilities should provide enhanced performance using only local communication and computation, including:

- *Autonomous patterned flight*. In many surveillance tasks, the RPVs must fly in formation to ensure complete coverage, maintain interaircraft distance, or present minimal radar return. This involves negotiating over task responsibilities, establishing communication protocols among the group, and

defining procedures for transitioning between formations or flight patterns in response to threats.

- *Data fusion among vehicles.* The different RPVs may be responsible for different portions of the intelligence-gathering process. This requires some means of representing hypotheses and confidence estimates, integrating new data, and deciding what information to send to others. Like Hearsay-II (Erman et al., 1981), the onboard systems will have to access multiple knowledge sources and maintain multiple lines of reasoning.
- *Cooperative planning and replanning.* The RPV fleet must react to contacts, altering the group's flight path to locate defenses and targets. The fleet members must also avoid dangerous terrain and weather and respond to threats. Such dynamic planning may be performed in a centralized manner by one member of the group (a leader), or it may be done by multiple group members, acting asynchronously and cooperatively. In either case, the planning will require the generation of maneuver options, simulation of the resulting trajectories (using whatever data are available), and evaluation of the projected partial solutions (Stefik et al., 1983).
- *Reconstitution after losses.* When RPVs are shot down or otherwise lost, the surviving vehicles must close ranks and determine new roles. Also, the vehicles must frequently reconstitute communication networks disrupted by jamming, noise, or damage. This requires polling group members, determining connection tables and capabilities, computing effective communication routings, and specifying new task assignments.

We produced a series of demonstration systems, described below, that exhibited many of the above capabilities. We did not pursue formal experiments with these systems, but we did examine many implementation options. In terms of the vocabulary introduced in Sec. II, the main distributed problem-solving tasks were formation keeping, data fusion,

and communications management. Conflict avoidance, the central task of ATC, was not of major importance here.

INITIAL RPV DEMONSTRATION SYSTEM: PATTERNED FLIGHT

We began our series of demonstration systems with the simplest problem: patterned flight over a benign environment. As shown in Fig. 5, this simulation involves three aircraft flying over a region without hostile defenses. The aircraft change coverage pattern (racetrack or figure 8), formation geometry (wave, vee, or stream), and spacing (close or wide) in response to command inputs. The coordination process involves several steps. The aircraft first determine if a change in leadership is necessary, and if so, they use a negotiation procedure much like that in our ATC implementation to select a new leader. Coordination is achieved by having the lead aircraft determine its own

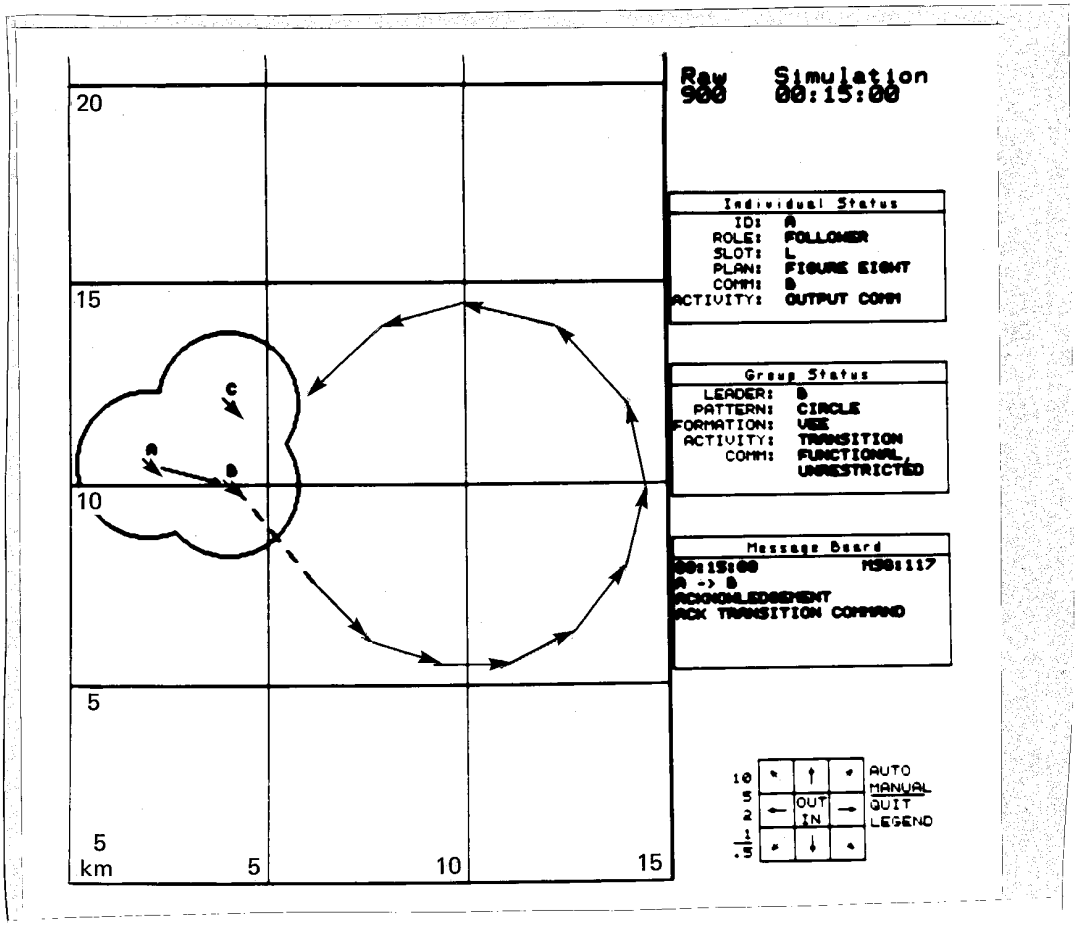

Fig. 5 -- Initial simulation of surveillance RPV coordination

trajectory and then send messages detailing its course and desired spacing to the other aircraft, who plan collision-free, minimum-time paths to match up with the leader. All messages in this initial system are error-free and have a free format structure. A typical message might be:

```
Message #112
sender: RPV1
recipient: RPV3
content: vee formation, wide spacing
type: point-to-point
time-stamp: 0400
```

The actual process of formation-keeping can be accomplished in two ways: (1) by maintaining a physical relationship with respect to the leader, or (2) by following a trajectory that should maintain group spacing. The first method requires frequent sensing or communication of leader position and execution of error-correcting responses. This tracking approach is appropriate if sensing and communication are reliable, course changes are frequent, or onboard navigation is difficult. The second option, relying on accurate trajectory-following, should require only occasional updates of leader position for confirmation of group spacing. In fact, if the aircraft can plan and follow their own path trajectories, they should be able to split off the group in response to threats or opportunities and rejoin later. This technique appears most appropriate in wide formations with few course changes or in situations with frequent threats and jamming. We implemented a combination of the two approaches in an object-oriented simulation. (The characteristics of the simulation are described on p. 42.) The follower RPVs used sensing and adjustment whenever possible but maintained their own trajectory plans.

Overall, this first demonstration system exhibited a rudimentary level of distribution. The several RPVs negotiated over roles, and each planned its own trajectory. We focused on leader-based behavior (in which the follower aircraft reacted to the leader's commands and keyed on his position), because this structure provided the simplest and most direct interactions. The alternative, an anarchic structure in which each aircraft can command any other, adds many complications, including deadlock and looping. In our scheme, a single leader was present, and leadership

changed from aircraft to aircraft according to the circumstances encountered. The leader assumed the bulk of planning responsibility, receiving information from and sending commands to the other aircraft. Each of our succeeding implementations relaxed the degree of centralization.

SECOND DEMONSTRATION SYSTEM: UNCERTAINTY REPRESENTATION

We next expanded the system to explore the problems of unreliable sensing and communication. As shown in Fig. 6, the aircraft fly over an environment with active defenses--command centers, ground control intercept (GCI) radar installations, and surface-to-air missile (SAM) sites. When necessary, the RPVs poll each other regarding status, sensing capabilities, and communication links. Each vehicle builds up its own uncertain model of the environment and fleet status, taking into account confidence degradations due to inaccurate sensing, communication

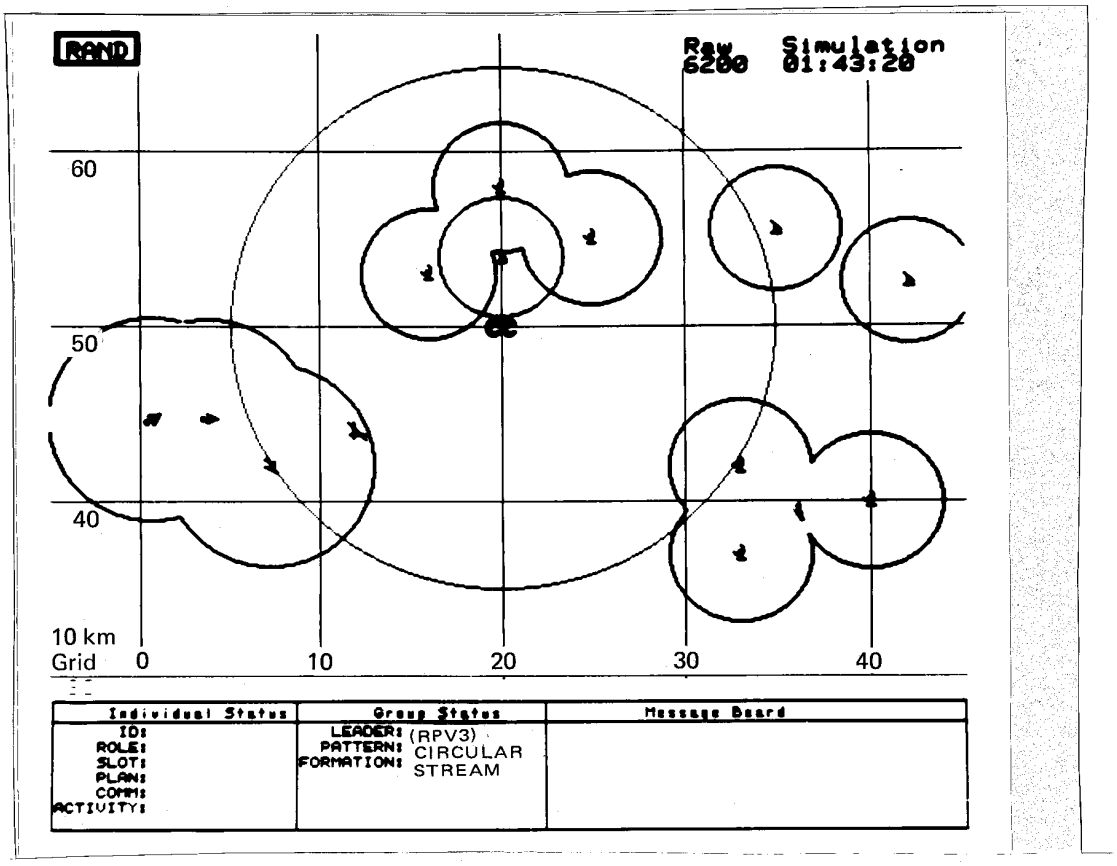

Fig. 6 -- Second implementation of surveillance RPV task simulation, with active defenses

noise, and incorrect intelligence. Database entries for defenses take the form of property lists with confidence factors, such as:

```
GCI site #4
   location: (45,34)
   status: active
   jammed: no
   confidence factor: 0.8
```

We used the uncertainty representation form of MYCIN (Shortliffe, 1976) to update each RPV's estimates of belief and disbelief in hypotheses about the defenses. This approach does not require the many prior and posterior probability estimates required by Bayesian analysis (Duda et al., 1979) or the many range estimates demanded by the Dempster-Shafer calculus (Garvey et al., 1981) or Inferno (Quinlin, 1982). Although the MYCIN approach has many limitations, primarily with respect to independence assumptions, we felt that the simplicity of form and ease of rule writing outweighed the possible errors. Such errors should be of minor significance in these early demonstration systems, because of the low frequency of updating and the coarseness of the behavioral responses.

The dynamics required for simulating sensing and detection are relatively straightforword. As shown in Fig. 6, the SAM and GCI sites have circular regions in which they can detect RPVs, and the RPVs have somewhat larger regions of defense sensing (they can passively sense radar emissions and therefore do not require a long reflection path). Sensing and detection in these regions are probabilistic, representing the effects of terrain, weather, ECM, and other factors. Updating is a sequential process--confidence in a hypothesis should increase if further contacts are made.

We introduced these effects in the simulation by adding false defenses, incorporating probabilistic sampling in the sensing/detection process, and corrupting some percentage of the communication messages. Figure 7 shows the sequence of events possible at each simulation update.

The sensing/detection cycle works in the following manner. At the beginning of an update, the system checks to see if any real or false defenses are within an RPV's sensing range. If a defense is within

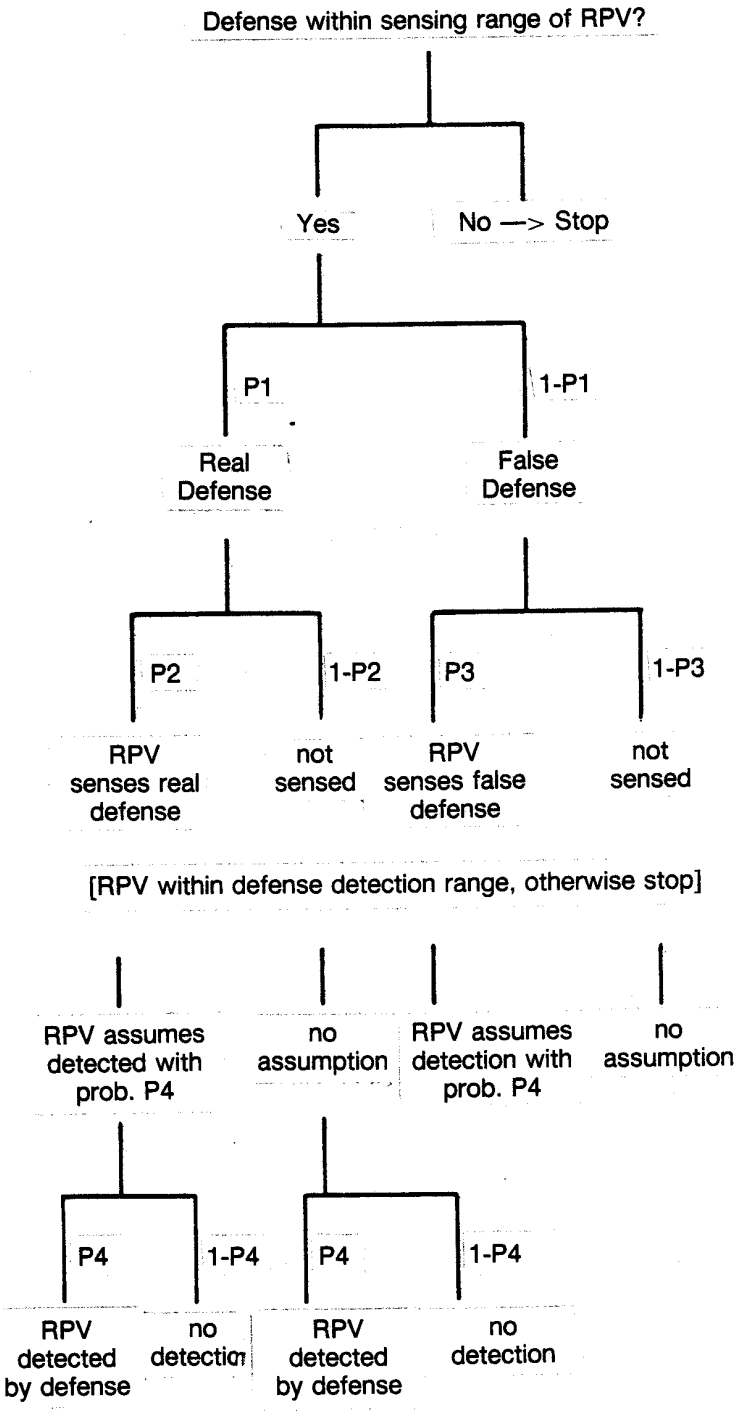

Fig. 7 -- Event tree for sensing and detection

range, it has a P1 likelihood of being real, and if real, a P2 likelihood of being sensed. False objects have a P3 chance of being sensed. If an RPV does sense a defense, the likelihood of that defense being real is

$$(P1 * P2) / ((P1 * P2) + ((1-P1) * P3)) \qquad [1]$$

The next layer in the event tree assumes the RPV is within detection range of a defensive site. If within range, the RPV believes it has a P4 probability of being detected. A randomization process uses the same P4 probability to send a message to the defense, if so indicated.

When an RPV senses a defense, it incorporates that information into its world-model. The sensing itself results in a measure of belief (MB) equal to the probability in Eq. [1]. The RPV checks to see if it already has an entry for the sensed defense type and location. If not, the RPV enters the MB directly into its database and sets the the confidence factor (CF) equal to it. If such an entry already exists, the RPV revises the measure of belief according to the following rule:

$$MB = MB1 + ((1-MB1) * MB2) \qquad [2]$$

where MB1 is the original entry and
MB2 results from the current sensing

There also may be disbelief present, represented as MD. Then the confidence factor CF = MB - MD. An MD can arise if a previously sensed defense is not again sensed (this indicates that the original contact may have been false). If this occurs, the probability of the defense being false is roughly

$$MD = (1-P1) / ((1-P1) + (P1 * (1-P2))) \qquad [3]$$

Communications further add to the uncertainty present by introducing a noise factor. The receiving RPV adjusts the original data

confidence by the probability of message corruption and updates its database entry in the same manner as in sensing.

With the introduction of degraded communications and unreliable sensing, we found that the role-negotiation process became more complex and the maneuver choices more important. The criteria used during leader negotiation expanded to include task knowledge (environment data and plans of others), current role, sensing region, communication-link strengths, and physical position. The RPVs also used these criteria to negotiate over data-fusion tasks, specifically, responsibility for GCIs and SAMs, for command centers, and for airfields.

The many formation, pattern, and spacing choices open to the group also had much more influence on task performance in this scenario than in the earlier perfect sensing and communication scenarios. For example, the wave formation with wide spacing provides the greatest coverage area but is most vulnerable to detection. The vee formation gives the shortest overall communication links but results in some penalty in coverage. The stream formation provides minimal radar return and easy following of the leader and, because of its narrow sensing area, is effective only for avoidance and strong secondary sensing confirmation. The following two rules illustrate some of the formation changes invoked by the leader in response to environmental conditions:

> If group is in stream or vee formation, and 20 seconds have passed without a contact or threat, change to wide wave formation
>
> If group is in wave or vee formation, and leader senses a SAM site with CF > 0.5, designate follower on opposite side to defense as leader and change to stream formation (this way the group may vector around the defense)

We also found that as the distributed problem-solving task became more complex, it became necessary to prioritize the functions of sensing, communication, planning, and control. We implemented a scheme similar to that developed in our ATC work. The following list shows the nomimal ordering of activities:

1. Input user message
2. Input command message
3. Plan trajectory in response to leader command
4. Execute action (if follower)
5. Perform sensing
6. Input data message
7. Announce leader
8. Input role-negotiation message
9. Send role-negotiation message
10. Plan formation change
11. Send commands
12. Send acknowledgment
13. Input acknowledgment

We attempted with this ordering to perform all activities preparatory to planning. In this way, the time-consuming leader planning operation would be assured of up-to-date information.

THIRD DEMONSTRATION SYSTEM: GENERAL SURVEILLANCE

Our final implementation, now in process, represents a general surveillance task in a hostile environment, with the RPVs avoiding defenses, sustaining losses, and regrouping. Here, communications are highly constrained because of the likelihood of detection and loss. The primary problems we encountered involve communication management, the user interface, and forms of logical inference.

One question that arises in communications management is information volunteering vs. information demanding. We examined these two options in the context of leader negotiation and communication-table updating. Information volunteering (in which the sender transmits data it expects the recipient needs) appears to result in fewer transmissions than a demand protocol, provided the sender has an accurate model of the recipient's needs. A comparison of volunteer and demand messages for a role-negotiation interaction is summarized in Fig. 8. In this example, we assume N RPVs are present. The minimum number of messages for each step is shown in parentheses at the right of the table. The demand form requires fewer messages (order N) than the volunteer form (order N**2) but tends to be time-consuming (at least one extra stage) and vulnerable to loss of the central agent.

Demand:		Messages
1)	Follower encounters problem, messages leader	(1)
2)	Current leader sends messages to followers requesting role factors	(N-1)
3)	Followers send role factors to leader	(N-1)
4)	Leader compares responses (unless incomplete) and sends messages announcing new leader	(N-1)
	Total:	3N-2
Volunteer:		
1)	Follower encounters problem, messages others with own role factor	(N-1)
2)	Other send role factors (except to initiator)	(N)(N-1)/2
3)	First to have all role factors announces leader	(N-1)
	Total:	$(N^2+3N-4)/2$

Fig. 8 -- Comparison of demand and volunteer protocols for communication

Communications management problems also appeared when we changed from three to five RPVs. In a larger fleet, the aircraft frequently do not have a direct transmission path to other aircraft. They have to route messages by the most direct and least loaded path, much like a packet radio system (Kahn et al., 1978). We considered three methods of routing: communication tables, route set-up packets, and spreading activation. Communication tables are built up by each RPV by listing which aircraft can communicate with which others. This requires the RPVs to indicate any changes in their links when they send a message, and to update their tables when they receive such an indication. The second method uses route set-up packets, special short-length messages sent prior to a multihop communication. When acknowledged, they provide a means for comparing the speed and noise of each possible path, but they tend to burden the channel. The third method, spreading activation, means that copies of a multihop message are sent along each possible pathway. This increases redundancy, virtually assures receipt, and

requires no table management overhead, but it can severely tax the channel capacity. We chose the communication table method in our work, because of the limited number of available agents and the high costs of communication.

RPV FLEET-CONTROL IMPLEMENTATION

We implemented the RPV simulation routines, the planning and problem-solving procedures, and some of the graphics facilities for the three demonstration systems in ROSS, an object-oriented programming language developed at Rand and written in Franz Lisp (McArthur and Klahr, 1982). Programs written in object-oriented languages consist of a set of objects that interact with each other via the transmission of messages. Each object has a set of attributes describing itself, and a set of message templates and associated behaviors. A behavior is invoked when an object receives a message matching the corresponding message template. A behavior is itself typically a set of message transmissions to other actors. In this fashion, ROSS and other object-oriented programming languages enforce a "message-passing" style of programming.

The ROSS programming style is well-suited to simulation in domains consisting of autonomous interacting objects. The style aids the understanding and modeling of distributed problem-solving systems, because objects can control their own activities through individual behaviors and maintain their own models of the world via their local databases. For example, data about sensed defenses are represented in the vehicle's database, and reactions to the defenses result from behaviors triggered by the receipt of messages. Our distributed fleet-control implementation can be thought of as consisting of three distinct types of processing: behaviors for simulating the scenario, behaviors for cooperative planning and control by the RPVs, and graphics behaviors for user display and interaction.

The simulation behaviors define aspects of the scenario and capabilities of the objects. Among these behaviors are defining trajectories, specifying time increments, calling randomization programs, sensing objects, and communicating messages. Special objects were defined for some of these functions. These processes were considered operational requirements rather than problem-solving activities.

The second type of processing consists of behaviors for distributed planning, coordination, and problem solving. Most of the activities described in the three demonstration sections fall into this category. Reasoning about role assignments, making decisions about coverage pattern, formation geometry, and vehicle spacing, and performing trajectory planning are included. Below we show the English and ROSS versions of a behavior for avoiding a sensed defense:

> If the group is in a stream formation, and the leader estimates its probability of detection by a SAM is greater than .6, then change to another coverage pattern.

```
(if (and
    (eq (~your formation) 'stream)
    (~you are leader))
    (greaterp (~your probability-of-detection of
              (~your sensed >SAM)) .6)
then
    (~you change coverage-pattern))
```

This example only hints at the fact that code in a ROSS simulation can be highly intelligible, modular, and modifiable. Most of the actions in the RPV simulation can be viewed as responses to messages and therefore are expressed naturally in this paradigm.

The graphics environment for our demonstration systems was programmed in a combination ROSS and C-based subsystem. Communication with the simulation objects was performed in ROSS, while device-dependent operations were implemented in C. The graphics subsystem was responsible for displaying task conditions, individual aircraft views, and rule firings. We found that these graphics capabilities, which are substantially more involved than those used in our ATC work, highlighted the user interface problems of portraying concurrent activities in a distributed system.

In some ways, the situation dynamics that had to be portrayed were more similar to those of a Time Warp mechanism (Jefferson and Sowizral, 1982) than those of a conventional distributed simulation, in which all objects step forward at the same rate. Each RPV tries to plan its route

for some distance into the future, and it is often at a different look-ahead time than its cohorts. If an RPV receives a message describing another RPV's plan, it checks for conflicts or constraints and, if necessary, backtracks to an earlier simulation time and replans. (This goes beyond Time Warps, as some actions are not virtual and cannot be rescinded.) The RPV then sends messages of its new plan to other affected RPVs. Tracking down chains of interactions can thus be quite involved. The user (who is already one frame behind in the animation) must trace backward and forward through all pertinent messages and resulting actions until the root cause is located.

The problem of simulating the maintenance and coordination of several independent databases also led us to consider implementing some of the functions using logic programming (specifically Prolog). Prolog can be particularly effective for database management, since it unifies the notions of data, rules, and queries (Kowalski, 1979). It derives answers to queries, using a very efficient inference procedure, and its basis in logic suggests its application for such inferential tasks as constraint satisfaction and data fusion.

Prolog clauses are of the form 'A if B1 and B2 and .. Bk', where each of 'A', 'B1'..'Bk' are conditions. The rule has an IF-THEN reading. If the right-hand side of a rule is empty, 'A' may be regarded as unconditionally true and hence as a piece of data. Thus if RPV1 has the following current state:

```
position      (30.0 40.0)
leader        rpv2
velocity      (600.0 0.0)
detected_by   radar1
```

RPV1 may represent this information by the following set of unconditional clauses:

```
position(rpv1,[30.0,40.0]) if ( )
leader(rpv1,rpv3) if ( )
velocity(rpv1,[600.0,0.0]) if ( )
detected_by(rpv1,radar1) if ( )
```

and RPV1 may use the same representation to express what it knows about its leader (RPV3):

```
position(rpv3,[20.0,25.0]) if ( )
velocity(rpv3,[550.0,100.0]) if ( )
detected_by(rpv3, radar3) if ( )
```

Rules are then easily added. If RPV1 is always constrained to keep a distance greater than 15 units from another RPV, it would have a rule of the following form:

```
properly_spaced(Vehicle1,Vehicle2)<-position(Vehicle1,P1) and
                    position(Vehicle2,P2) and
                    distance(P1,P2,D) and
                    D>15.
```

If RPV1 wishes to determine whether it is properly spaced with respect to its leader, it would execute the query,

```
leader(rpv1,L) and properlyspaced(rpv1,L).
```

In this case, it will execute successfully, since the distance between RPV1 and RPV3 is approximately 18.

Prolog's inferencing capabilities are expected to be useful for such query answering, for verifying consequences of actions, and for reducing communications. Consequence verification may be performed by checking whether each consequence (desirable or undesirable) is logically implied by its present model of the world and the contemplated action. Communications may be reduced in a similar way. An RPV would attempt to infer whether another RPV knows a certain piece of information before transmitting it on. Conversely, an RPV would attempt to infer a piece of information from its own database before explicitly querying another RPV.

The advantage of using Prolog for database management (and more generally for inferencing) instead of LISP is that Prolog provides a simple, unified framework for representing data, rules, and inferencing. We are in the process of implementing a Prolog based in LISP, so that any Prolog programs will easily interface with our existing programs. We anticipate that the existence of logic programming primitives in the ROSS system will considerably increase ROSS's expressive power.

IX. CONCLUSIONS AND FUTURE WORK

Our approach to distributed problem solving has been primarily an empirical one, using several forms of simulation to explore key aspects of agent interaction--task negotiation, communication management, cooperative planning, and reorganization. Our initial work in air-traffic control, for example, has shown the importance of being able to interrupt, suspend, and resume activities within each agent. The ATC work also pointed out the sensitivity of system behavior and performance to changes in organization and information-distribution policies, particularly with respect to leadership decisions. Our subsequent simulation of surveillance RPV fleet control demonstrated the effectiveness of object-oriented programming for simulating and displaying behaviors of multiple interacting objects with common goals. The system was able to illustrate the flow of data and the evolution of responsibility as conditions changed. The system also lent itself well to the problem of local uncertainty representation in a highly probabilistic environment. We are now augmenting the ROSS object-oriented system with activity prioritization and logical-inference functions.

The ATC and RPV work has concentrated on functions involving cooperation among *autonomous* vehicles, principally collision avoidance, patterned flight, and surveillance of a hostile environment. We plan to extend these functions in the near future to involve *shared control*, in which a human operator acts in a supervisory role or takes over direct control of one of the aircraft. We also plan to widen the scope of actions to include defense suppression, decoy operations, special attack maneuvers, and damage assessment. Invoking these added options, the RPV fleet would transition frequently between organizational forms, altering its communication networks and "regrowing" connections following each operational phase. Protocols for such transitions are expected to be much more complex than those developed for the basic functions of formation keeping, data fusion, and avoidance responses.

Another major goal is to demonstrate a fully distributed system in which the agents take actions in a heterarchic, asynchronous fashion. Such a cooperative structure might involve frequent negotiation over tasks, burst communications whenever possible to minimize database disparities, and plan backtracking in response to commands and constraints sent by the other group members. We expect that such a heterarchical organization will primarily be useful in situations of extreme duress--radio silence, heavy jamming, and high attrition. Our next step will be to modify the simulation to examine the performance of such an organization.

One of the more pervasive problems we encountered is that of the user interface in a distributed system--producing a window on the workings of the many separate agents. We noted that certain displays appeared to be essential: textual displays of activities being performed by each agent, graphic displays of situation assessments (with interaircraft disparities highlighted), and animated graphic displays showing how each aircraft's plans and assumptions play out over time and space. At the same time, the user needs the control over such conventional functions as pan, zoom, time stepping, and level of detail. Development of an appropriate user environment for observing or participating in the distributed problem-solving process should occupy researchers well after the mechanics of interagent communication and planning are solved.

REFERENCES

Appelt, D. E., *Planning Natural-Language Utterances to Satisfy Multiple Goals*, Technical Note 259, SRI International, 1982.

Dalkey, N. C., *Group Decision Making*, Report UCLA-ENG-7749, School of Applied Science, University of California, Los Angeles, July 1977.

Davis, R., "A Model for Planning in a Multi-agent Environment: Steps Toward Principles for Teamwork," Working Paper, MIT Artificial Intelligence Laboratory, Cambridge, 1981.

Davis, R. and R. G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving," Memo 624, MIT Artificial Intelligence Laboratory, Cambridge, 1981.

Duda, R. O., J. G. Gaschnig, and P. E. Hart, "Model Design in the PROSPECTOR Consultant System for Mineral Exploration," in D. Michie (ed.), *Expert Systems in the Micro-electronic Age*, Edinburgh University Press, Edinburgh, 1979, pp. 153-167.

Erman, L. D., P. E. London, and S. F. Fikas, "The Design and an Example Use of HEARSAY-III," *IJCAI*, Vol. 7, 1981, pp. 409-415.

Fahlman, S., "A Planning System for Robot Construction Tasks," *Artificial Intelligence*, 5(1), 1974, pp. 1-49.

Fikes, R. E. and N. J. Nilsson, "Strips: A New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence*, 2(2), 1971, pp. 189-208.

Forgy, C. L., "The OPS5 Users Manual," Technical Rept. CMU-CS-79-132, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, 1981.

Garvey, T., J. Lowrance, and M. Fischler, "An Inference Technique for Integrating Knowledge from Disparate Sources," *IJCAI*, Vol. 7, 1981, pp. 319-325.

Goldberg, A. and A. Kay, "Smalltalk-72 Instruction Manual," Report SSL 76-6, Xerox PARC, Palo Alto, 1976.

Gray, C. M., K. D. Rehm, and D. R. Woods, "The Drone Formation Control System," *Military Science*, 1982, pp. 11-26.

Hyman, A., "Where are the RPVs?" *Aerospace International*, 17(3), July-August 1981, pp. 40-44.

Jefferson, D. and H. Sowizral, *Fast Concurrent Simulation Using the Time Warp Mechanism, Part I: Local Control*, The Rand Corporation, N-1906-AF, December 1982.

Kahn, R., S. Gronemeyer, J. Burchfiel, and R. Kunzelman, "Advances in Packet Radio Technology," *Proceedings of the IEEE*, 66(11), November 1978, pp. 1468-1496.

Konolige, K., "A First Order Formalization of Knowledge and Action for a Multi-agent Planning System," *Machine Intelligence 10*, 1981.

Kowalski, R., "Algorithm = Logic + Control," *Communications of the ACM*, 22(7), July 1979.

Kunz, J. C., R. J. Fallat, D. H. McClung, J. J. Osborne, R. A. Votteri, H. P. Nii, J. S. Aikins, L. M. Fagan, and E. A. Feigenbaum, "A Physiological Rule-based System for Interpreting Pulmonary Function Test Results," Report HPP-78-19, Heuristic Programming Project, Computer Science Dept., Stanford University, 1978.

Lesser, V. R., S. Reed, and J. Pavlin, "Quantifying and Simulating the Behavior of Knowledge-based Interpretation Systems," *Proceedings of the First Annual National Conference on Artificial Intelligence*, Stanford University, 1980, pp. 111-115.

Lesser, V., *A High-level Simulation Testbed for Cooperative Problem Solving*, COINS Technical Report 81-16, University of Massachusetts, Amherst, 1981.

McArthur, D. A. and P. Klahr, *The ROSS Language Manual*, The Rand Corporation, N-1854-AF, May 1982.

McArthur, D. A., R. Steeb, and S. Cammarata, "A Framework for Distributed Problem Solving," *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, 1982, pp. 181-184.

Newell, A. and H. Simon, "GPS--A Program that Simulates Human Thought," in E. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963.

Newell, A. and H. Simon, *Human Problem Solving*, Prentice-Hall, New York, 1972.

Nii, H. P. and N. Aiello, "AGE (Attempt to Generalize): A Knowledge-based Program for Building Knowledge-based Programs," *IJCAI*, Vol. 6, 1979, pp. 645-655.

Quinlin, R., *Inferno: A Cautious Approach to Uncertain Inference*, The Rand Corporation, N-1898-RC, September 1982.

Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, 5(2), 1974, pp. 115-135.

Sacerdoti, E., *A Structure for Plans and Behavior*, Elsevier North-Holland, New York, 1977.

Shortliffe, E. H., *Computer-based Medical Consultation: MYCIN*, American Elsevier, New York, 1976.

Smith, B. A., "Israeli Use Bolsters Interest in Mini-RPVs," *Aviation Week and Space Technology,* July 18, 1983, pp. 67-71.

Smith, R. G., "A Framework for Problem Solving in a Distributed Processing Environment," STAN-CS-78-700, Stanford University, 1978.

Steeb, R., S. Cammarata, F. A. Hayes-Roth, P. W. Thorndyke, and R. B. Wesson, *Distributed Intelligence for Air Fleet Control*, The Rand Corporation, R-2728-ARPA, 1981.

Stefik, M., " Planning with Constraints (MOLGEN: Part 2)," *Artificial Intelligence*, Vol. 16, 1981, pp. 141-169.

Stefik, M., J. Aikins, R. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti, "The Architecture of Expert Systems," in F. Hayes-Roth, D. Waterman, and D. Lenat (eds.), *Building Expert Systems*, Addison-Wesley, Reading, Pa., 1983.

Sussman, G., *A Computational Model of Skill Acquisition*, American Elsevier, New York, 1975.

van Melle, W., "A Domain-independent Production-rule System for Consultation Programs," *IJCAI*, Vol. 6, 1979, pp. 923-925.